Terminology and an Architecture on Data Independence

The described architecture involves two independent concepts related to logical data structure and physical data organization to implement dynamic data independence. Data independence is a property of a data system which permits variations in the representation and organization of data as stored without requiring reprogramming. It also permits variations on the program's view of the data without requiring rearranging the data. Static data independence is provided by an implementation which requires that if a change to the stored data need be made, then new data descriptors must be written, all existing data must be modified to conform to the new data descriptors, and all programs which use this data recompiled to conform to the new data descriptors. Dynamic data independence provided by an implementation in which the system consults the data descriptors as it traverses its file, thus permitting variations along the file. This permits partially rearranged data and permits different performance factors or priorities to be applied to different portions of a file.

The first concept is a set of definitions which clearly identifies the interests of the various parties in a data base environment, and identifies the interfaces. The second concept is an architecture of tables, control blocks and descriptors associated with a quantity of stored data. These tables include, essentially, descriptors of data storage, an index to this data and to the associated descriptors, an intermediate descriptor which is used in both name resolution and mapping algorithm, a program work area descriptor and a mechanism for naming the particular logical record required. These descriptors are unbound and are associable in many-to-many relationship. The mapping of the stored records onto the physical records is a function of a data organization and not this architecture. Thus, this architecture is applicable to providing data independence on any data set organization. The following definitions apply. An entity record is the conceptual collection of all information, for which a representation can be constructed or recorded in a data base about an entity, that is, data which is logically related. A logical record is a collection of one or more fields as viewed by an application program. By definition, all of the fields in a logical record represent information about the same entity. Thus, it is a subset of an entity record. Each entity and thus each logical record has one unique identifier. A collection of logical records comprise a file. A stored record is a collection of related data elements upon a medium. These relationships depend on the storage organization, and may be independent of the logical relationships of the various entity records about which information is stored. A collection of stored records comprises a data set. A physical record is a unit of recording upon a medium such as between gaps on tape, or between address markers or one sector on disk. A collection of physical records comprises a subdivision of a volume.

The following describes the tables used in this architecture as seen in Fig. 1. A source program data declaration compiles or is interpreted as a logical record descriptor 1, which contains the names and representations of the fields of data as the object programs will require them. The logical record descriptor is "compiled" into a system interface. An entity record descriptor 3 contains the names of all of the potential fields which may be required in any object program in the family of applications requiring this data, or maintained for real-time

browsing or anticipated queries, and the source of each of the fields. To identify the instance of which stored record(s) include the data element(s) in involved in each field, the entity record descriptor 3 also includes the algorithm or transformation of what the user presents as the logical record identifier (actually the entity record identifier) to stored record(s) identifier. It acts as the mapping director for stored to logical transformation. There is no "format" associated with an entity record descriptor. The format of the data as stored is in a stored record descriptor. The format of the data as seen by an object program is in a logical record descriptor. However, a default attribute descriptor can be associated with an entity record for the purpose of display. The stored record descriptor 5 contains the name of each data element in the stored record, and the representation in storage. The stored record index 7 contains the identifier for each stored record, a pointer to its location in storage, and a pointer to the descriptor applicable to the particular stored record. In this most general case, the association of each stored record with its descriptor is necessary to support dynamic variation. The pointers to the descriptors may be factored, in that a data set label could identify the extents over which a given set of descriptors apply, or the descriptors may apply to an entire data set. The pointer to the descriptor may be in the stored record rather than the index. Self-defining fields of data may be substituted for descriptors.

The flow of control information and data is seen in Fig. 2. The name and version of the entity record set, the entity identifier and the logical record descriptor is the interface between the object program and data management. This interface is interpretively processed at each data transaction. Data management determines by matching the names of the fields with the names of the attributes how to retrieve each field. The names of the required elementary items, and the names and version of the required stored records which correspond to the entity identifiers are developed. Through the indexes the stored records and their associated descriptors are retrieved. The sapping implied by the element descriptor and field descriptor is accomplished. The fields are materialized and the logical record returned to the object program, or data elements are materialized and the stored records returned to storage.

A many-to-many association of tables is possible. A logical record is associated with one entity record. However, many logical records may be associated with one entity record. One entity record may be associated with several stored records - either by the substitution of data sets or by partitioning of data among data sets, or both. One stored record may be associated with several entity records.

In a smaller data management implementation a reduction in interface may be made, resulting in less interpretation and less function. The pointers to the descriptors may be removed from the index and may be factored for the entire data set, resulting in loss of dynamic data independence. The stored record descriptor may be written as a source language descriptor and copied into the source program during compilation. This conceptually makes the entity record congruent to the stored record, makes the logical record a subset of the whole entity record, and also deletes interpretive mapping. Interpretive binding can be provided. For example, a compiled mapping procedure generated at an explicit or implicit OPEN, or at a specific request, would remain valid as long as the

descriptors remain valid. Similarly, the data element descriptors could be involved with the object program and the pertinent manipulative instructions incrementally compiled.

While it is not an integral part of the preceding control architecture, it is assumed that logical record descriptors are written by an application programmer in his source code; the entity record descriptor and the stored record descriptor written by a data base administrator in some specialized control language; and the indexes constructed by data management as directed by the data base administrator.

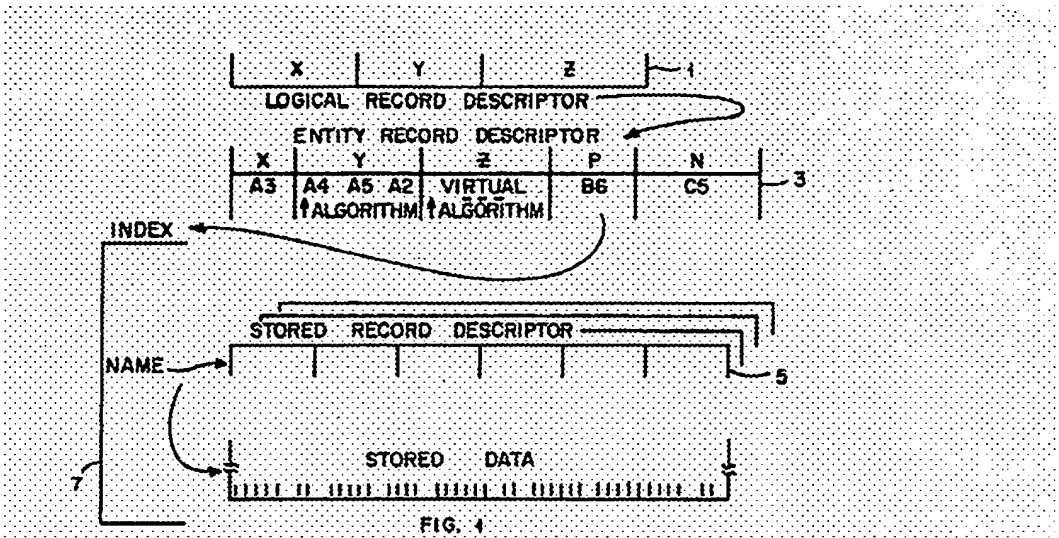
X
Y
Z
1
LOGICAL RECORD DESCRIPTOR
ENTITY RECORD DESCRIPTOR
X
Y
Z
P
N
A3
A4 A5 A2
VIRTUAL
B6
C5
3
ALGORITHM
ALGORITHM
INDEX
STORED RECORD DESCRIPTOR
NAME
5
STORED DATA
7

FIG. 1

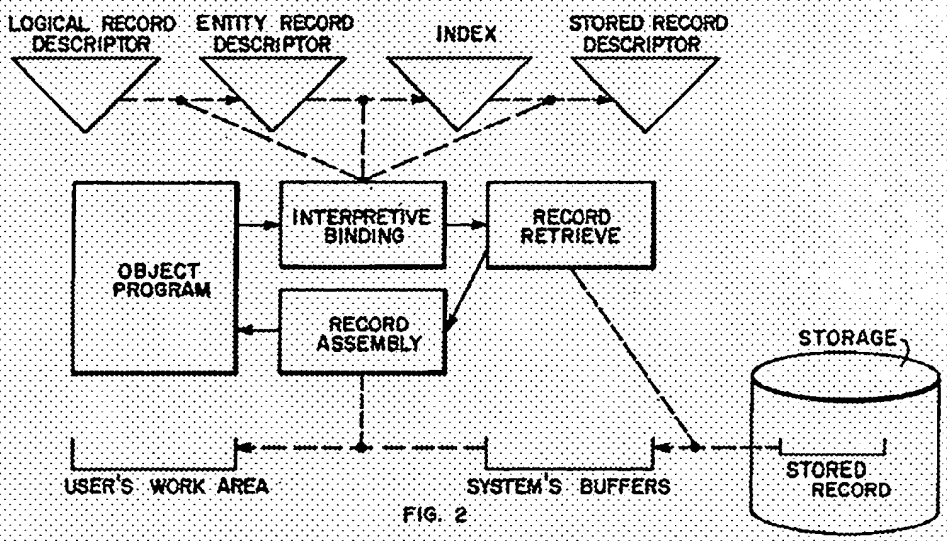
LOGICAL RECORD DESCRIPTOR
ENTITY RECORD DESCRIPTOR
INDEX
STORED RECORD DESCRIPTOR
OBJECT PROGRAM
INTERPRETIVE BINDING
RECORD RETRIEVE
RECORD ASSEMBLY
STORAGE
USER'S WORK AREA
SYSTEM'S BUFFERS
STORED RECORD

FIG. 2